An Empirical Analysis of Equivalence Partitioning, Boundary Value Analysis and Random Testing

Stuart C. Reid
Cranfield University
Royal Military College of Science,
Shrivenham, Swindon, Wilts SN6 8LA, UK

Abstract

An experiment comparing the effectiveness of equivalence partitioning (EP), boundary value analysis (BVA), and random testing was performed, based on an operational avionics system of approximately 20,000 lines of Ada code

The paper introduces an experimental methodology that considers all possible input values that satisfy a test technique and all possible input values that would cause a module to fail, (rather than arbitrarily chosen values from these sets) to determine absolute values for the effectiveness for each test technique.

As expected, an implementation of BVA was found to be most effective, with neither EP nor random testing half as effective. The random testing results were surprising, requiring just 8 test cases per module to equal the effectiveness of EP, although somewhere in the region of 50,000 random test cases were required to equal the effectiveness of BVA.

1. Introduction

Many National, International, and Industry Standards mandate the use of specific testing techniques and test adequacy criteria, but what is the rationale for their choice of techniques? In many cases the choice seems arbitrary - in some even contradictory - and even when an informed approach is attempted there is a lack of solid evidence on which to base the decision.

Research on the fault-finding capability of testing techniques has been ongoing for more than twenty years, yet, despite this, there is still no consensus on the most effective techniques. In recent years much of this work has been theoretical, or in support of little-used, or -known, techniques, which, although valuable, provides little help to the practising software engineer (or writer of standards). Several papers have advocated empirical studies of testing techniques, such as [1], [2], [3], [4], and [5]; this paper reports on such a study. The effectiveness of the two most popular black box testing techniques used in the module testing phase of software development, equivalence partitioning (EP) and boundary value analysis (BVA), are compared with each other, and with random testing. The paper shall also provide guidance on the most effective practical application of these techniques.

One of the difficulties with experiments and case studies of test effectiveness is in the choice of truly representative test cases. Normally the 'testers' in such studies arbitrarily choose test cases from large sets of inputs that satisfy the requirements of the technique under investigation (such as equivalence partitions for EP), but their choice will not necessarily be representative of other testers. This can make the experimental results rather 'hit and miss', unless a statistically large enough number of testers and test cases are used. The methodology used in this study avoided this potential problem by considering *every* input that satisfied the technique under investigation (for instance, every possible value in an equivalence partition). In this way an absolute measure of test effectiveness for given modules and testing techniques was generated.

The hypotheses that formed the basis for the study were that BVA was more effective than EP, and that both these techniques were more effective than random testing. Further, more refined hypotheses, based on variants of the techniques, are described later in the paper.

The next two sections describe the testing techniques used in the study and the inherent difficulties in experimenting on test effectiveness. Section 4 considers related work on determining test effectiveness. Section 5 describes the methodology used in the study, while sections 6 and 7, present the results and the techniques used to determine their statistical significance. Section 8 discusses the results and, based on them, provides guidance on the application of the techniques studied. Finally con-

0-8186-8093-8/97 $10.00 © 1997 IEEE

clusions and references are presented in sections 9 and 10.

2. Background

All systematic module testing techniques (except random testing), are based on partitioning the input domain of the module. These techniques aim to produce partitions that exercise a particular feature of the module, such as branches in the code, or boundary values as described by the module specification. These techniques are based, in part, on the assumption that all inputs in a partition will be treated similarly by the module. That is, if one input value from a partition causes a module to fail, then it is assumed that the remaining inputs from the partition will also cause the module to fail, and vice versa for inputs that do not cause failure. If this is the case, then the partitions are considered to be homogeneous, or truly revealing [5].

It follows from this that these partitioning schemes, or partition testing techniques, must produce distinct partitions of the input domain and that none may overlap, as any intersections could not be considered homogeneous - values within these intersections would not behave similarly to *both* partitions. However, in practice, most of the testing techniques do not meet the criterion of homogeneity, but instead produce overlapping partitions, and so are known more correctly as subdomain testing techniques; EP and BVA, in practice, fall into this category.

The definitions of the testing techniques used in this study are taken from the BCS Software Component Testing Standard [6]. This standard includes definitions of both test case design techniques and test measures; for the rest of this paper these will simply be referred to as testing techniques as it is assumed that the test case design technique is applied to achieve 100% coverage of the corresponding test measure. The definition of EP in [6] requires the use of subdomains rather than true partitions, stating that "a test case may exercise any number of partitions". However, to maintain consistency with the definitions in [6], the term 'equivalence partition' will be used in this paper when describing both true partitions and subdomains in the context of EP.

The Standard provides definitions of EP and BVA that are open to interpretation, and it explains, in the guidelines clause of the standard, that there are "a spectrum of approaches" represented, at its extremes, by the "one-to-one" and "minimised" approaches. An aim of this study has been to determine which approach is more effective, and so, for EP and BVA, both approaches were considered.

Using either approach, the equivalence partitions/boundary values are first identified from the specification, and then test cases are derived to exercise them. For the 'one-to-one' approach a separate test case is created corresponding to each identified equivalence partition/boundary value, while any remaining input parameters, not used to define the equivalence partition/boundary value, may take any valid value that allows the equivalence partition/boundary to be exercised. For the 'minimised' approach the minimum number of test cases is generated that will exercise all identified equivalence partitions/boundary values, so a single test case may well exercise several equivalence partitions/boundary values for different input parameters; in fact, with this approach, test cases are generated that exercise as many equivalence partitions/boundary values as possible. Where different combinations of inputs satisfy the requirement of achieving a minimal test case set then one is chosen at random.

The difference in the two approaches can be seen in the following example for EP. Consider the following fragment of a simple module specification:

if Y_{IN} greater than 4 do P
if Y_{IN} less than 3 do Q
if Y_{IN} any other value do R
if abs(X_{IN}) less than 1 do S

Let figure 1 represents the domain of all possible inputs to the module; in this example there are two input parameters, both real numbers, one represented as values on a horizontal axis (X_{IN}, in the range -4 to +4), with the other represented as values on a vertical axis (Y_{IN}, in the range 0 to 6). The input domain is divided into four equivalence partitions, EP1, EP2, EP3 and EP4, with the first three running horizontally, and the fourth running vertically. It is easy to see from the figure that these are not *true* partitions as they overlap (so are actually subdomains - there are six *true* partitions).

If using the 'one-to-one' approach then the four identified equivalence partitions would lead to four test cases, whose inputs satisfy the following:

$-4 \le X_{IN} \le 4$ AND $4 < Y_{IN} \le 6$ EP1
$-4 \le X_{IN} \le 4$ AND $3 \le Y_{IN} \le 4$ EP2
$-4 \le X_{IN} \le 4$ AND $0 \le Y_{IN} < 3$ EP3
$-1 < X_{IN} < 1$ AND $0 \le Y_{IN} \le 6$ EP4

For the 'minimised' approach, the minimum number of test cases that can cover all four equivalence partitions is required. In this case there are three, which would be drawn from the sets represented by EP1∧EP4, EP2∧EP4, and EP3∧EP4:

$-1 < X_{IN} < 1$ AND $4 < Y_{IN} \le 6$ EP1∧EP4
$-1 < X_{IN} < 1$ AND $3 \le Y_{IN} \le 4$ EP2∧EP4
$-1 < X_{IN} < 1$ AND $0 \le Y_{IN} < 3$ EP3∧EP4

In this example no alternative combinations are possible that create a minimal set.

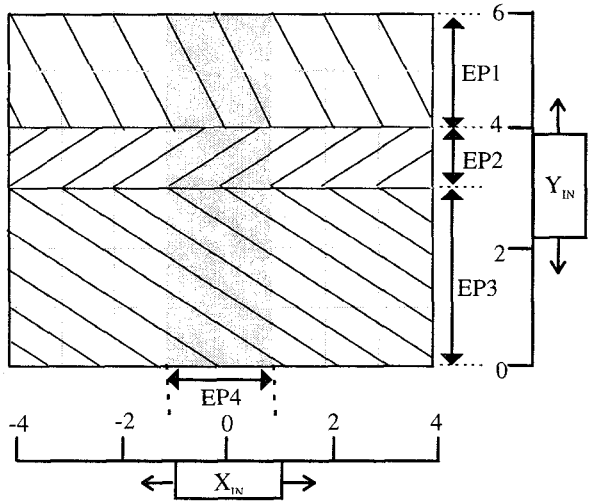

Figure 1: Equivalence partitions

In common with most module testing, no operational profile of inputs was available for the modules studied and so a uniform distribution was used for the generation of inputs for random testing.

3. Experimental difficulties

Test effectiveness cannot be measured directly, instead experimenters use surrogate measures, known as response variables, such as the number of defects detected, or, in this case, the probability of detecting a fault. A number of other factors that affect the experimental results, but are not measured, are known as state variables, which characterise the experiment. The results should be applicable elsewhere *given the state variables are similar.* Attempting to manage this 'correspondence' does, however, raise a number of questions:

- are the experimental testers working at the same skill level as real testers?
- if only a small number of test cases are chosen from a subdomain in an experiment can they be representative of all values in it?
- can seeded faults be used, or are only real faults worthy of study?
- are the experimental module specifications of typical quality and notation?
- are experiments language-specific, or can results be applied to other languages?
- are the experimental test techniques well-defined and repeatable?

Only a few of the numerous factors that need to be considered in experiments of this kind have been presented here. The full set of state variables that characterise the experiment described in this paper are provided in Appendix A, and reasons for their choice are given in section 5.

4. Related work

A number of papers have been published in this area, covering both theoretical ([3], [4], [10], [5], [11], [12], [22], [23]), and empirical studies ([13], [14], [7], [8], [15], [16], [2], [17], [9]). Perhaps one of the surprising features of these studies is that the empirical studies tend to pre-date the theoretical work. There is still not enough data available to validate much of the theoretical work and a software tester reading the empirical studies would find that there is little consensus on which testing techniques are the most effective.

Although there have been many studies on the effectiveness of generic partition testing techniques, there has been little study of specific techniques, such as EP and BVA. Many of these studies into generic partition testing techniques have also assumed true partitioning (i.e. no overlapping), whereas the practical application of most techniques tends to generates overlapping subdomains ([2], [4]). Random testing has fared much better, perhaps because it is much easier to generate test inputs. The results of [14], that random testing can be cost effective for many programs, subsequently validated by [5], have also generated interest in this technique, as intuitively it should be far less effective than systematic techniques.

Future empirical studies must provide sufficient details to enable their results to be compared and used to build up a cohesive body of knowledge in the area. A parametric framework has been suggested [1], which would allow experiments to be classified and so both compared and repeated.

5. Experimental methodology

This section initially presents the hypotheses that the study intended to assert, and then describes the experiment and the theory behind it. The experiment comprised five stages: Failure Analysis, Creation of Fault Finding Sets, Creation of Test Case Sets, Derivation of Probabilities, and finally Analysis of Results. All but the last stage, which is described later, are described in this section, and are shown in figure 2.

In this research the fault-finding sets were initially derived (see 5.3) from the faults identified by a failure analysis of the system (see 5.2). The subdomains re-

quired to satisfy test coverage criteria were then identified and defined as the test case sets (see 5.4). Both the fault-finding sets and test case sets were defined using techniques borrowed from symbolic execution. Then, using the concepts underlying domain testing, a definitive probability of detection is calculated in terms of 'overlapping' N-space convex polyhedra, which correspond to the test case sets and the fault-finding sets respectively (see 5.5).

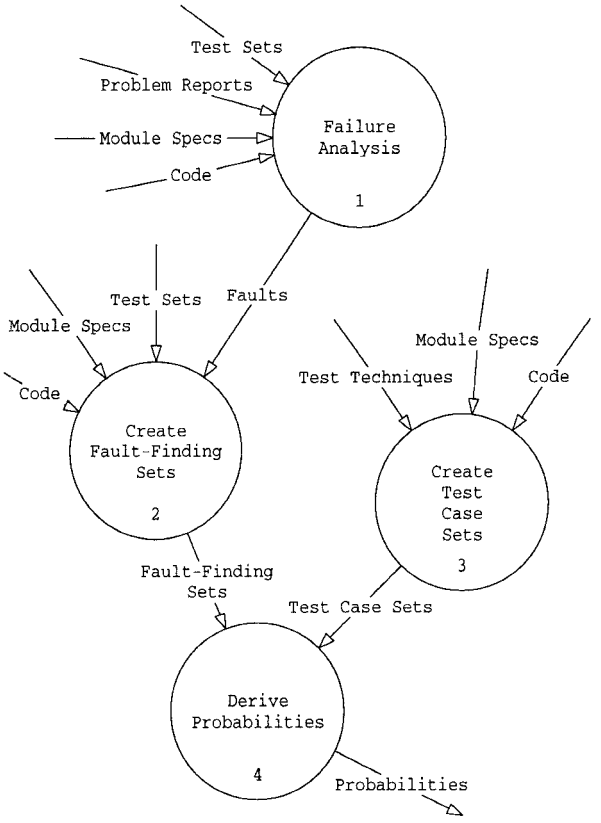

Figure 2: Experimental methodology

5.1. Hypotheses

The experiment considered five pairwise comparisons. In each case the hypothesis asserted was that the first technique in each pair was more effective than the second.

- BVA (one-to-one) with EP (one-to-one)
- EP (one-to-one) with Minimised EP
- BVA (one-to-one) with Minimised BVA
- EP (one-to-one) with Random
- BVA (one-to-one) with Random

For a fair comparison, this was based on random testing being performed using the same number of test cases as required by the technique it was being compared against. For instance, if 'EP (one-to-one)' of a particular module required twelve test cases then the probability of detection for the random testing of that module was based on twelve randomly-generated test cases.

5.2. Failure analysis

A project was chosen where the system was already in operation, and a failure analysis was used to identify the faults in the system that could have been detected by module testing. Problem reports were the main source of failure information, as they were required to initiate any change to the system. The reported faults were analysed using the original and modified source code, design specifications and test specifications.

The system has now been flying for some time and the lack of change requests from the customer suggested that *most* of the faults (that would cause failures) had been documented, and so were included in this failure analysis. The complete system was analysed, which meant that all relevant faults were identified. By this means no experimental bias was introduced by choosing a subset, and it was hoped that a typical and complete cross-section of faults was considered.

Problem reports had to be raised for any change to the code or documentation *after* it was lodged with configuration control. This meant that the faults identified were those left after the developers had carried out their informal module testing, which, for the case study project, was branch testing. So any faults detected by this branch testing were not included in the analysis. An average branch coverage of 99.0% was achieved by the developers, the last 1.0% being due to failures in test execution, rather than unreachable or missed code. This 'loss' of data on faults found by informal testing was not considered a handicap as the faults that were *left* were considered the most important. Hetzel [18] states that the "defects of greatest interest are those still in the product after release, not those already discovered."

The faults that could have been detected by module testing were a small subset of all faults found with the system (less than 10%). They were then used to create the fault-finding sets. The full results from the failure analysis of the system can be found in [19].

5.3. Creation of fault finding sets

Once a fault was identified and understood, the set of inputs to the module that would cause the fault to be exposed were determined and these were used to create fault-finding sets (there will normally only be a single set corresponding to a given fault, however in some cases

disjoint sets may expose a single fault). An example is shown in Figure 3, with two fault-finding sets shown.

These fault-finding sets take the form of simultaneous equalities and inequalities on the module inputs representing the input conditions for exposing the fault. These fault-finding sets bear a strong resemblance to the system of equalities and inequalities used in symbolic execution systems. In the example the fault-finding sets (FF1 and FF2) could be described by:

$-4 \le X_{IN} \le 0$ AND $1 \le Y_{IN} \le 4$ FF1
$2 \le X_{IN} \le 4$ AND $1 \le Y_{IN} \le 5$ FF2

where the two input parameters are X_{IN} and Y_{IN}, as shown in figure 3.

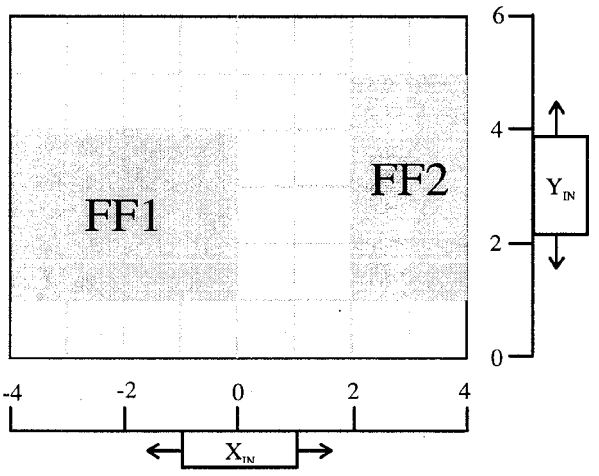

Figure 3: Fault-finding sets

5.4. Creation of test case sets

The test case sets were all derived by the same person from the definitions of the testing techniques described in section 2, with the aim of minimising the effect of the tester skill level on the experiment.

The test case design techniques considered correspond to test coverage criteria in that 'complete' use of the test case design technique will generate a test case suite that will achieve 100% coverage of the test coverage criterion. Test case sets were constructed that corresponded to those input conditions that had to be satisfied to achieve the given test coverage criterion.

As with the fault-finding sets, the test case sets take the form of simultaneous equalities and inequalities on the module inputs. For a given test coverage criterion there will be a number of test case sets, with each set corresponding to one of the subdomains of the module's input domain required to be exercised by the criterion. For instance, the path coverage criterion for a typical module would lead to a large number of subdomains - one for each of the distinct paths through the module.

Using the subdomains shown in figure 1 (EP1 to EP4) then the following four test cases sets would be created if using an EP 'one-to-one' approach:

$-4 \le X_{IN} \le 4$ AND $4 < Y_{IN} \le 6$ TC1
$-4 \le X_{IN} \le 4$ AND $3 \le Y_{IN} \le 4$ TC2
$-4 \le X_{IN} \le 4$ AND $0 \le Y_{IN} < 3$ TC3
$-1 < X_{IN} < 1$ AND $0 \le Y_{IN} \le 6$ TC4

5.5. Derivation of probabilities

The fault-finding capability of a technique for a particular module is derived by measuring the degree of intersection between the fault-finding sets and the test case sets. The domains of both the fault-finding sets and the test case sets can be considered to be convex polyhedra in N-space, where N is the number of input parameters to the module. The probability of a fault being detected is measured by the 'overlap' between the fault-finding polyhedra and the test case polyhedra. In general the probability of detection can be calculated as follows:

Let D denote the module's input domain, of size d, (thus d=|D|), m points of which cause the module to fail. Then these m points shall define the extent of the fault-finding sets, and if FF represents the union of all fault-finding sets, then m=|FF|.

If P_{DET} is the probability that a randomly-chosen input, based on a uniform distribution, will cause a failure (and so detect a fault) then $P_{DET} = m/d$.

If TC represents a test case set then the probability of the fault being detected by a single randomly-chosen test case from TC is given by $P_{DET}(TC, FF) = m_{TC}/d_{TC}$ where d_{TC}=|TC| and m_{TC}=|FF∧TC|.

Thus, the probability of missing the fault is

$$P_{MISS}(TC, FF) = 1 - P_{DET}(TC, FF) = 1 - m_{TC}/d_{TC}.$$

Then, given that n test case sets (TC1,TC2,..,TCn) are required to satisfy the test coverage criterion, C, the probability of the fault (represented by the fault-finding sets FF) being missed is the product of the miss probabilities for each of the n test case sets, thus:

$$P_{MISS}(C, FF) = \prod_{i=1}^{n} P_{MISS}(TCi, FF).$$

And so, the probability of detecting the fault is:

$$P_{DET}(C, FF) = 1 - P_{MISS}(C, FF) = 1 - \prod_{i=1}^{n} (1 - P_{DET}(TCi, FF)).$$

For random testing, the input domain (D) for the module can be considered as a single test case set, but, as stated

earlier in section 5.1, different numbers of random test cases were used to provide 'fair' comparisons with the other techniques.

Thus the earlier formulae, which provide probabilities based on a single test case per subdomain (test case set) can be modified for random testing by considering D as a single test case set, with n test cases. Thus,

$$P_{DET}(RANDOM, FF) = 1 - (1 - P_{DET}(D, FF))^n .$$

This assumes that test cases are selected with replacement. As the number of test cases is very small when compared with the domain sizes this should have minimal effect, and if random inputs were generated automatically this would probably be the case anyway.

An example calculation can be shown by considering the fault-finding sets (FF1 and FF2) and the test case sets (TC1 to TC4) described earlier in sections 5.3 and 5.4, respectively, and shown in figures 1 and 3. From a consideration of the overlap of the fault-finding sets and the test case sets, the probabilities of detection can be determined for each of the four test case sets, thus:

$$P_{DET(TC1)} = 1/8 \Rightarrow P_{MISS(TC1)} = 7/8$$

$$P_{DET(TC2)} = 3/4 \Rightarrow P_{MISS(TC2)} = 1/4$$

$$P_{DET(TC3)} = 1/2 \Rightarrow P_{MISS(TC3)} = 1/2$$

$$P_{DET(TC4)} = 1/4 \Rightarrow P_{MISS(TC4)} = 3/4$$

Given that one test case is used per test case set then the probability of all four missing the fault is the product of the probability of missing for each of the individual test case sets:

$$P_{MISS} = 7/8 * 1/4 * 1/2 * 3/4 = 21/256 = 0.082, \text{ or } 8.2\%.$$

So, the overall probability of detection,

$P_{DET} = (1 - 0.082) = 0.918$, or 91.8%.

6. Results

Table I contains the probabilities of detecting the faults for each of the different testing techniques considered for each of the 17 modules. Each module contained a single fault, identified in the failure analysis. Where a value of one appears in the table then if the specified test technique had been applied to the given module it would have been certain to detect the fault. Conversely, values of zero imply that application of the test technique would never have discovered the fault. All intermediate values represent a probability of detecting the known fault.

The 'mean' value in the table can be interpreted as the average probability of detecting a fault in a module (in this sample) by the use of the given technique. The 'sum' can be interpreted as the average number of faults that would have been found across all 17 modules by the use of the given technique.

TABLE I : Probabilities of detection

Module ID	Random (1)	EP (1 to 1)	Minimised EP	Random (#EP)	BVA (1 to 1)	Minimised BVA	Random (#BVA)
1	0.0000171042	0.000153	0.000055	0.000205231	0.000421	0.72	0.0007523085
2	0.0000342085	0.000379	0.00017413	0.000410425	0.00106045	0.72	0.00150407
3	0.0000342085	0.00021111	0.0001085	0.000205233	0.00044791	0	0.00082068
4	0.00001534	0.0000705	0.0000215	0.0000920364	0.000204	0	0.0003680951
5	0.0000000101	0.00003	0.00001529	0.000000061	1	1	0.0000002034
6	0.03515625	0.1787	0.09375	0.1333815	0.3843	0	0.372027
7	0.125	0.578125	0.578125	0.41382	1	1	0.65639
8	0.3804065	0.9076	0.85086	0.9434228	1	1	0.999888
9	0.0000868055	0.00052074	0.00052074	0.000347177	1	1	0.00208125
10	0.397	1	1	0.98252	1	1	1
11	0.401	1	1	0.99	1	1	1
12	0.000015259	0.000095497	0.00006499895	0.0001525795	1	1	0.000396658
13	0.0000076294	0.00004826	0.00003322	0.0000762914	1	1	0.0001983455
14	0.0000868055	0.00052074	0.00052074	0.000347177	1	1	0.000694233
15	0.0060763888	0.06261	0.0276	0.05913	0.98686	0.9961	0.192104
16	0.21875	0.95551	0.75	0.9153	0.999997	0.9961	0.9998
17	0.401	1	1	0.99	1	1	1
Mean	**0.12**	**0.33**	**0.31**	**0.32**	**0.73**	**0.79**	**0.37**
Sum	**2.0**	**5.7**	**5.3**	**5.4**	**12.4**	**13.4**	**6.2**

'Random (1)', 'Random (#EP)' and 'Random (#BVA)' correspond to random testing with a single test case, the number of test cases required to achieve EP (one-to-one), and the number of test cases required to achieve BVA (one-to-one) respectively.

Table II presents the number of test cases required for EP and BVA (both 'one-to-one' and 'minimised'), along with the mean for each of them.

TABLE II : Number of test cases

Module ID	EP (1 to 1)	Minimised EP	BVA (1 to 1)	Minimised BVA
1	12	6	44	14
2	12	6	44	14
3	6	3	24	10
4	6	3	24	10
5	6	3	20	12
6	4	3	13	5
7	4	4	8	5
8	6	5	19	14
9	4	4	24	5
10	8	4	24	13
11	9	4	26	13
12	10	9	26	23
13	10	9	26	23
14	4	4	8	5
15	10	6	35	26
16	10	6	35	26
17	9	4	26	13
Mean	**7.6**	**4.9**	**25.1**	**13.6**

7. Statistical analysis

Statistical measures were applied to the five pairs of testing techniques (as listed in section 5.1) using the probability of detection of faults as the measure of test effectiveness.

Hypothesis testing was used to determine whether the different techniques provided any significant increase in test effectiveness. The null hypothesis in each case was that there was no real difference in the effectiveness of the two techniques, while the alternate hypothesis was that one technique was significantly better.

Two separate statistical tests were used to assert the null hypothesis, due to the unknown distribution of the fault detecting probabilities. Initially a technique based on a normal distribution was used (the correlated form of Student's t-test [20]), the results of which can be seen in Table III, labelled 't'. As tests for population means are generally insensitive to departures from normality, this was considered reasonable, however a second set of tests were performed, that are applicable whether the distribution is normal or not, to confirm the results. The Mann-Whitney form of Wilcoxon's test with correlated samples [20] was used, the results of which can also be seen in Table III, labelled 'U'.

In each of the comparisons the value P_R was the risk of being wrong if the null hypothesis was rejected. Thus, ideally, the values calculated for P_R should have been as low as possible, so that the null hypothesis could be rejected in favour of the alternate hypothesis (that one technique *was* more effective than the other). Values of P_R greater than 0.05 for t-test and 0.1 for Wilcoxon are labelled as "insig." to indicate that no significant difference between the two samples was detected.

TABLE III : t-test and Wilcoxon results

Technique Pairs	P_R(t)	P_R(U)
BVA (1:1)-EP (1:1)	<0.005	0.075
EP (1:1)-Minim. EP	0.05	0.025
BVA (1:1)-Minim. BVA	insig.	insig.
EP (1:1)-Random	insig.	0.05
BVA (1:1)-Random	<0.005	0.065

Both of the statistical tests were applied using their correlated form, which is applicable when there is correspondence between the i_{th} member of each sample under test. This meant that the probabilities of detection were considered in pairs, with the specific faults and modules as the connecting factor. This correlation technique allows the specific effects that are common to the pair to be minimised.

8. Discussion

8.1. Comparison of techniques

Each of the pairwise comparisons was considered in turn, followed by branch testing, the baseline technique that was used by the developers before submitting code to configuration control and thus prior to the study.

8.1.1. BVA (one-to-one) with EP (one-to-one). For none of the faults did EP have a higher probability of detection than BVA, and the mean of the probabilities for BVA was more than twice that for EP. Both statistical techniques supported the hypothesis that BVA was more effective than EP. BVA therefore appeared to be far more effective than EP, however, to achieve this higher level of effectiveness more than three times as many test cases were required.

Despite the extra test cases BVA appears far more attractive to the practitioner. On average, nearly seven

extra faults would have been found by using BVA and the difference in the number of required test cases does not reflect a proportionate increase in the cost of applying BVA over EP. The major cost in applying *either* technique is in identifying the equivalence partitions (from which the boundaries are derived for BVA). The extra cost introduced by using BVA is mainly in the generation of the expected results for the additional test cases, as, by using automated test tools, the execution and checking of tests will not incur any significant extra costs.

8.1.2. EP (one-to-one) with Minimised EP. For none of the faults did Minimised EP have better probabilities of detection than its one-to-one counterpart, but overall there was only a 0.02 difference in the mean probabilities (equivalent to a 6% improvement). The statistical techniques applied considered the risk of being wrong in rejecting the null hypothesis at 5% or less. On average the one-to-one approach required 50% more test cases than the minimised approach, but this was less than 3 test cases per module overall.

The small cost of including the extra test cases required for the one-to-one approach appears to reap only a small reward in a barely increased mean probability of detection. There is a slight overhead, however, in determining the minimal test set from the identified equivalence partitions when using the minimised approach. Overall it is difficult to recommend one approach ahead of the other.

8.1.3. BVA (one-to-one) with Minimised BVA. Neither statistical measure provided any basis for rejecting the null hypothesis (that there is no real difference between these techniques). The difference between the mean probabilities for the techniques was just 0.06 (equivalent to 8%), however there was a considerable difference in the number of test cases required by the two approaches. The one-to-one approach required nearly twice as many as the minimised approach, which, on average, was 11.5 extra test cases. This, and the slightly better mean probability of detection, appear to give the minimised approach the edge over the one-to-one approach.

When considering the values for probability of detection for the minimised approach, two outliers were noted (ID 1 and 2 in Table I) with much greater values than for any of the other techniques. These were due to faults that would only be revealed if two of the input parameters took specific boundary conditions simultaneously. The minimised approach to BVA generates test cases that include boundary values for each input parameter when possible, which is why the probability is so much better in this case. The minimised approach to BVA thus focuses testing on the vertices of the notional N-space convex polyhedra that represents the module's input domain by targeting the intersection of edges at a vertex. Less specifically, the one-to-one approach focuses on whole edges of planes of the polyhedra.

Problems with the minimised approach, however, are that it cannot hit each vertex (more than a million possible input boundary combinations was not uncommon in the modules investigated), and, also, some faults did not 'hide' in the corners, but were found in the middle of an edge. Examples of this 'blindness' can be seen in the zero probabilities for modules/faults with ID 3, 4 and 6. This appears to confirm the belief that reliance on a single technique can be dangerous, and that complementary techniques need to be identified and used.

8.1.4. EP (one-to-one) with Random. The statistical results indicated that it was difficult to differentiate between these two techniques (based on this sample) and the mean of the probabilities differed by only 0.01. The average number of tests required to satisfy EP was 7.6, but by using 8 tests per module, random testing achieved the same level of effectiveness as EP.

Random testing requires much less effort than EP to generate the test input values, and, with practically no extra test cases required to achieve parity of test effectiveness with EP, then random testing compares very favourably.

The poor performance of EP is due to the distribution of faults within the equivalence 'partitions'. The technique identifies subdomains that must be exercised, which are generally of unequal size. If the fault has a low probability of detection and lies in one of the larger subdomains then the technique will perform poorly compared to random testing. With truly revealing subdomains then EP performs much better than random testing, but, as can be seen in Table I, this was the case for only three of the faults.

Compared to random testing, EP *should* provide more efficient test coverage of the program logic (as long as it matches the logic in the module specification). This targeting of test effort will normally be effective for faults in program logic, however, in this study the modules had already been informally tested to achieve 100% branch coverage. Thus many of the faults that could be expected to be found by EP had been removed already, contributing to its relatively poor performance. This appears to indicate that a test strategy that requires both EP and branch testing is not particularly effective.

8.1.5. BVA (one-to-one) with Random. According to the mean values for probability of detection of faults then

BVA was clearly far more effective than random testing, and the statistical measures appear to confirm this.

Due to the diminishing returns provided by increasing the number of random test cases then without automated test oracles random testing cannot compete with BVA. Approximately 35,000 random tests per module are required to achieve the same mean probability as BVA (one-to-one) and 50,000 for minimised BVA.

8.1.6. Branch testing. This had been carried out by the developers before the study began, and 100% coverage had been achieved for each of the modules studied that had a fault. As this level of testing is the mainstay of several organisations and standards then it is worth noting that it left several faults that appeared relatively easy to spot (a single randomly-chosen test case for each module would, on average, have detected two of the seventeen faults and three of the modules contained truly revealing equivalence partitions).

8.2. Limitations of the experiment

The use of the concepts underlying domain testing and symbolic execution to define the fault-finding and test case sets means that the study has been restricted in a similar manner to these techniques. For instance, only modules with linear predicates could be analysed, however all the modules studied satisfied these constraints, a situation reflected in symbolic execution studies [21]. One way of surmounting possible problems in future studies may be to use automatic random test input generation to produce large numbers of inputs from within each test case set and to subsequently measure the proportion that cause the module to fail. The same technique could be used to validate the analytically-determined probabilities calculated from the 'overlap' of the polyhedra used in this study.

It is recognised that the identification of equivalence partitions, which is required for both EP and BVA, can be somewhat subjective. This could cause some experimental bias, but the high level of detail provided by the module specifications and the use of well-defined procedures should have mitigated this risk.

The assumption that each of the test techniques would be applied to achieve 100% coverage will not hold for all projects. The author doubts, however, that the setting of test coverage criteria of less than 100% coverage for EP and BVA is practical. All equivalence partitions and boundaries must be identified to calculate the coverage achieved by testing a subset and there would be little justification for then not testing to 100%.

9. Conclusions

This paper has presented the results of a study into the test effectiveness of a number of black box testing techniques. The experimental methodology used here has differed from previous experiments as the derived test effectiveness is based on an analysis of *all* inputs that satisfy the test technique rather than arbitrarily chosen inputs from this set.

The results, in summary, are:

- BVA was the most effective technique studied, achieving a highest mean probability of detection of 0.79, compared with 0.33 for EP. However, to achieve this, nearly twice as many test cases were required (13.6 for BVA , 7.6 for EP).
- For lower levels of effectiveness, random testing appears to have the advantage over EP. Just eight random test cases per module were required to achieve the same level of effectiveness as EP, and random testing will also be less expensive to perform as no test case design is required.
- Although random testing appears effective for achieving lower levels of test effectiveness, it requires a prohibitive number of test cases (50,000) to reach the levels achievable by BVA.
- Minimised BVA focuses testing more on the extremes of the input domain than one-to-one BVA; the study found little to choose between the two approaches.
- Minimised EP was consistently slightly less effective than one-to-one EP, although it required marginally fewer test cases. Again, there was little to choose between the two approaches.
- Using complementary techniques (rather than a single technique) appears sensible, and, if choosing a black box technique to complement branch testing, then BVA appears to be more effective than either EP, or random testing.

Obviously the results of this study have taken little account of cost, except in general terms; measures could not be taken as the techniques were not used to detect faults, but rather to create test case sets. More study of the costs of applying testing techniques needs to be performed, although the relatively large differences in test effectiveness results presented here may well overshadow relatively small differences in the cost of their application.

A number of the faults were difficult to detect using any of the techniques studied (see Table I). A number of white box testing techniques, supposedly more effective than branch testing, warrant investigation and it is the intention to reuse the experimental methodology with these techniques to determine their relative effectiveness.

The next step will then be to re-apply the methodology to other systems to determine if the results are system-specific.

As with all such experiments the results are characterised by the conditions under which it took place (described in Appendix A) and their extrapolation to other environments is not implied.

10. References

[1] Roper, M. et al, 'Towards the Experimental Evaluation of Software Testing Techniques, *Proc. EuroSTAR '94*, Brussels, Oct. 1994.

[2] Frankl, P.G. and Weiss, S.N., 'An Experimental Comparison of the Effectiveness of Branch Testing and Data Flow Testing' *IEEE Trans. on Software Eng.*, Vol. 19, No. 8, pp. 774-787, Aug. 1993.

[3] Chan, F. T., Chen, T.Y., Mak, I.K. and Yu, Y.T., 'Practical considerations in using partition testing', *Proc. SQM '94*, Edinburgh, July 1994.

[4] Weyuker, E.J. and Jeng, B., 'Analyzing Partition Testing Strategies', *IEEE Trans. on Software Eng.*, Vol. 17, No. 7, July 1991, pp 703-711.

[5] Hamlet, R. and Taylor, R., 'Partition Testing Does Not Inspire Confidence', *IEEE Trans. on Software Eng.*, Vol. 16, No. 12, pp. 1402-1411, Dec. 1990.

[6] 'BCS Software Component Testing Standard', Issue 3.3, April 1997 (available from the author).

[7] Basili, V.R. and Selby, R.W., 'Comparing the Effectiveness of Software Testing Strategies' *IEEE Trans. on Software Eng.*, Vol.SE-13, No.12, pp. 1278-1296, Dec. 1987.

[8] Myers, G.J., 'A Controlled Experiment in Program Testing and Code Walkthroughs/Inspections' *Comms. of the ACM*, Vol. 21, No. 9, pp. 760-768, 1978.

[9] Vouk, M.A., Tai, K.C. and Paradkar, A., 'Empirical Studies of Predicate-Based Software Testing', *Proc. Fifth International Symposium on Software Reliability Eng.*, Monterey, Nov. 1994.

[10] Jeng, B., and Weyuker, E.J., 'Some Observations on Partition Testing', Proc. *ACM SIGSOFT 3rd Symposium on Software Testing, Analysis, and Verification*, ACM Press, pp. 38-47, Dec 1989.

[11] Chen, T.Y. and Yu, Y.T., 'On the Expected Number of Failures Detected by Subdomain Testing and Random Testing', *IEEE Trans. on Software Eng.*, Vol. 22, No. 2, pp. 109-119, Feb. 1996.

[12] Hamlet, R., Theoretical Comparison of Testing Methods' *SIGSOFT Software Eng. Notes*, Vol. 14, Iss.8, pp. 28-37, 1989.

[13] Ntafos, S.C., 'A Comparison of Some Structural Testing Strategies', *IEEE Trans. on Software Eng.*, Vol. 14, No.6, pp. 868-874, June 1988.

[14] Duran, J.W. and Ntafos, S.C., 'An Evaluation of Random Testing', *IEEE Trans. on Software Eng.*, Vol. SE-10, No. 4, pp. 438-444, July 1984.

[15] Weyuker, E.J., 'More Experience with Data Flow Testing', *IEEE Trans. on Software Eng.*, Vol. 19, No. 9, pp. 912-919, Sep. 1993.

[16] Selby, R.W., 'Combining Software Testing Strategies: An Empirical Evaluation', *Proc. of the Workshop on Software Testing*, Canada, 15-17 July 1986.

[17] Girgis, M.R. and Woodward, M.R., 'An Experimental Comparison of the Error Exposing Ability of Program Testing Criteria', *Proc. of the Workshop on Software Testing*, Canada, 15-17 July 1986.

[18] Hetzel, W.C., 'Making Software Measurement Work', *QED Publishing Group*, 1993.

[19] Reid, S.C., 'Test Effectiveness in Software Module Testing', *Proc. EuroSTAR '94*, Brussels, Oct. 1994.

[20] Cooper, B.E., 'Statistics for Experimentalists', *Pergamon Press*, 1969.

[21] Coward, P. D., 'Symbolic Execution Systems - A Review', *Software Eng. Journal*, Vol. 3, Part 6, pp. 229-239, 1988.

[22] Weyuker, E.J. and Frankl, P.G., 'A Formal Analysis of the Fault-Detecting Ability of Testing Methods', *IEEE Trans. on Software Eng.*, Vol. 19, No. 3, pp. 202-213, Mar. 1993.

[23] Weyuker, E.J. and Frankl, P.G., 'Provable Improvements on Branch Testing', *IEEE Trans. on Software Eng.*, Vol. 19, No. 10, pp. 962-975, Oct. 1993.

Appendix A - Characterising the study

The chosen case study project was a military aircraft display system using Motorola 68020 and Intel 80186 processors, developed to DO178A level 2 (essential). The final system comprised approximately 20,000 lines of Ada code (executable statements) and 3,500 lines of assembler. Six software engineers worked on the project, with between 2 and 14 years experience, at an average of 6½ years and development took nearly two years.

The notation used for the module specifications was structured english. The level of abstraction of the specifications was generally low, just above the source code.

The implementation language was Ada 83. The project coding standard mandated that neither GOTO statements nor tasking should be used and subprograms should not exceed 200 lines of code. The average number for lines of code per module studied was 78.5.

The project required that functional testing was carried out and a number of test coverage levels be achieved, although from discussion with the testers and from project documentation it became apparent that functional testing was not performed. Statement and branch coverage levels of 100% were required, while LCSAJ coverage to 60% was required, but achieved by default.